1 Introduction

Before we may proceed to the solution of the problem we shall make some observations. Firstly, let us consider when a line in the page of Jack's notebook is impossible for a given number of pilgrims and a given number of euros in Jack's pocket:

Claim 1 *A **COLLECT** line is always possible. A **PAY** line is possible only if the amount to pay is no larger than the amount of cash Jack currently has. A **IN** line or **OUT** line is only possible if the current amount of cash is divisible by the number of people in the group before this line and (in case of the **OUT** line) if the number of people to leave is smaller than the number of people in the group (Jack has to stay).*

Claim 2 *Consider a sequence S of lines in Jack's notebook. Suppose that before the first of these lines was written there were k pilgrims and m euros and after the lines were written the pilgrimage has k' pilgrims and m' euros. Then if the same lines were written with k pilgrims and $m + kx$ euros at the beginning, then after the writing of these lines the pilgrimage would have k' pilgrims and $m' + k'x$ euros — adding money obviously does not change the number of pilgrims in the pilgrimage, and the claim about money is a simple inductive fact.*

Thus we may note that adding a multiple of the number of pilgrims k in euros does not spoil the possibility of k being a solution for a given page. Thus we may assume Jack has enough money for all the **PAY** lines on the page at the beginning of the page, thus we may assume a **PAY** line is always possible. In particular, we have the following observations:

Observation 3 *Any **COLLECT** lines on the page may be ignored.*

Indeed, the **COLLECT** lines only add a multiple of the current number of pilgrims to Jack's pocket. Due to Claim 2 this amount per person will simply carry over from line to line, not affecting the possibility of **IN** or **OUT** lines occuring.

Observation 4 *Any initial sequence of **PAY** lines may be ignored.*

The initial **PAY** lines are always possible. Their net effect is to substract some amount e of cash from Jack's pocket. If the remaining part of the page is possible with k pilgrims and m euros, then the whole page is possible with k pilgrims and $m + e$ euros.

Observation 5 *Any final sequence of **PAY** lines may be ignored.*

Indeed, whatever amount of money was paid at the end Jack did not have to deal with fractional number of euros since no one came in nor out afterwise.

Finally, to check whether a sequence of lines is possible we shall use the following claim:

Claim 6 *Suppose that part of our input looks like that: **IN**/**OUT** a_1 **PAY** k_1 **PAY** k_2 ... **PAY** k_n **IN**/**OUT** a_2 and after a_1 pilgrims joined (or left) the trip there could have been exactly x people in the group. Then we have that x is a divisor of $k_1+k_2+\ldots+k_n$. Moreover, any x of such property can be number of people in the group after line **PAY** k_n.*

Indeed, if it was not so when a_2 people join or leave the group Jack would have to deal with fractional numbers of euros, which is guaranteed to have never occured.

Observation 7 *If input consists of no **PAY** lines except at the beginning and end (thus only **IN** and **OUT** lines) and $\Delta p_1, \Delta p_2, \ldots, \Delta p_n$ is a sequence of changes of number of people in the group (Δp_i corresponds to i-th line: **IN** Δp_i or **OUT** $-\Delta p_i$) then any number greater or equal to $min_k - (\Delta p_1 + \Delta p_2 + \ldots + \Delta p_k)$ can be the number of pilgrims at the beginning of the page.*

Having considered all the observations and claims we may propose a scheme of the problem solution:

1. Remove all **COLLECT** lines.
2. Remove the initial and ending sequence of **PAY** lines (if they occur).
3. If (after the preprocessing) there are no **PAY** lines compute the minimal number of pilgrims basing on observation 7.
4. In the other case we have input consisting of interlacing sequences of **IN/OUT** and **PAY** lines.
5. Take the first sequence of **PAY** lines, compute its sum S of values paid.
6. For every divisor of S check whether it can be a number of people in the remaining input sequence.

We need to remember to compute all the divisors of S in no longer than $O(\sqrt{S})$ time.

The last dragon left to slay is checking whether a number l can be the number of people in the remaining input sequence. This can be done by a single loop over the sequence. We will remember the number of people k if at the beginning there were l pilgrims. This value will need to be updated after each **IN/OUT** line. For each block of **PAY** lines we shall compute its sum and if it is divisible by the number of people m is a solution and it is not in the other case.

2 Model solution

Model solution (file *pil.{cpp/java}*)is an implementation of the above algorithm. Firstly input is filtered respectively to points 1) and 2). Later on, the call of *no_pays()* fuction handles the case when input consists no **PAY** lines. Then the main program function is called. It sums the values of the first remaining block of **PAY** lines (call this value sum), and for each pair of its divisors j and sum/j it adds them to result if they are possible numbers of people in the group. This is done by calling the function *possible()*, having ensured that the considered value is greater or equal to $1+$ adjustment respective to number of people joined/left in preceeding **IN/OUT** lines. The function *possible()* implements the idea described at the end of the previous section.

Let N be the size of input. Time complexity of the model solution is $O(N\sqrt{2000N})$ and memory complexity is $O(N)$.

3 Slow solution 1

Slow solution 1 (*pils1.{cpp/java}*) is very similar to the model solution. The only difference is that it computes all the divisors of S simply by looping over all numbers lesser than S, which takes significantly more time. Let N be the size of input. The time complexity of the model solution is $O(2000N^2)$ and memory complexity is $O(N)$

4 Tests

Test cases consists of files:

- *pil0.in* — example test,
- *pil1.in* — simple correctness tests,
- *pil2.in* — more sophisticated correctness tests.
- *pil3.in* — effectiveness tests consisting of short (1-3 element) **IN/OUT** initial and ending blocks and a large (40-48 element) block of **PAY**s with respectively large amounts of money (1850-2000). Thus sum of money is sufficiently large to make the slow solution run significantly longer.
- *pil4.in* — random effectiveness tests. A test case consists of interlacing blocks of **IN/OUT** (that are usually short) lines and **PAY** lines. First significant block of **PAY**s is quite long so that the number whose all divisors need to be computed grows on average to $50,000$-$70,000$.

The first two files were written manually, others are created automatically by program *pilingen.cpp* for the reason being that effectiveness tests should consist of around $10,000$ cases, which makes for a total of $400,000$-$500,000$ lines. Since **COLLECT** lines are irrelevant for the problem solution only a few of them are put randomly in last two test files.